スクリプト言語による計算数論システムの開発

AUTHOR(S):

松井, 鉄史

CITATION:

松井, 鉄史. スクリプト言語による計算数論システムの開発. 数理解析研究所講究録 2004, 1395: 144-149

ISSUE DATE:

2004-10

URL:

http://hdl.handle.net/2433/25940

数理解析研究所講究録 1395 巻 2004 年 144-149

スクリプト言語による計算数論システムの開発

松井鉄史

MATSUI TETSUSHI

東京都立大学大学院理学研究科

DEPARTMENT OF MATHEMATICS, TOKYO METROPOLITAN UNIVERSITY*

Abstract

スクリプト言語の一つである Python によって計算数論システムを開発する構想の基本概念を説明し得失を議論する。

1 開発の動機及び意義

数論的計算を主な対象とする計算システムである計算数論システムには H. Cohen が中心になって開発が進められてきた PARI や M. Pohst が中心になって開発が進められてきた KANT あるいは P. Zimmer が中心になって開発が進められ 2002 年から東京都立大学の中村ら [1] に開発が引き継がれた SIMATH などがある。筆者もこの引き継がれた SIMATH の開発に参加している。今回のスクリプト言語による計算数論システムの開発の構想はこの開発の経験に基づくものである。

1.1 SIMATH における問題

我々は、SIMATH の開発の中でいくつかの問題に気付いた。

SIMATH 自身は C 言語で書かれているため、メモリー管理を自らの手で行なわなければならない。具体的には、関数定義に現れる変数及び引数について、そのポインターをガベージコレクションから守るために一つ一つ登録しなければならない。[1)]確かに、`malloc` や `free` を直接使わなくていいようにはなっているが、それらを隠蔽する部分は SIMATH の一部であり、場合によっては手を加えなければならない。

また、このメモリー管理で管理される構造は単精度整数のリストに限定される。多倍長整数や楕円曲線もリストで表現されるなど実質的にリストだけで全てが書かれるため、その限定は特に制限事項には見えないかも知れない。しかし、保守・改良などソースコードを読む立場に立つと、型の情報が欠落するため非常に理解しにくいコードになる。

さらに、[1] に述べられているように SIMATH を 64 ビット機で動作するような改良を行なったが、このようなことが必要だったのはもともとのソースコードが 32 ビット CPU だけを動作対象とするように記述されていたからである。動作対象を限定することでプログラムの作成は容易になるが、結果的に移植作業を誘発し、その利益を享受することより苦労を強いられることの方が多いだろう。

*tetsushi@tnt.math.metro-u.ac.jp

1) コード記述上は変数を登録する形で書き、プリプロセッサによってポインターに変換する。

最後になるが、ユーザーが使うのは今まで述べてきたような C 言語ではなく simcalc という対話式インタープリターが多いだろう。すなわち、システム記述言語とユーザーの使う言語が分離しており、ユーザーの経験をシステムに反映することは単純にはできない。また、このような独自のインタープリターを実装・保守することはあまり数学に関係のないタスクであり、しなくて済むものならしないで済ませたい。

これらの問題点を解消するために、スクリプト言語の採用を検討する。次節でスクリプト言語、特に Python について概観し、次々節で問題点がどのように解消されるか見る。

1.2 Python

Python はスクリプト言語の一種に数えられる。スクリプト言語という言葉には明確な定義はないが、一般的にインタープリターとして実装される言語の一群で、容易に使えるところに特徴がある。最も一般的に使われるものに CGI 言語として名を馳せた Perl や、教育用言語に起源を持つ Python、あるいは日本人が創始者であり特に日本で人気のある Ruby などがある。

1.2.1 スクリプト言語の特徴

スクリプト言語は一般的にインタープリターとして実装されるプログラミング言語で、使い易さを目指して開発されているような言語である。使い易さの実現にはさまざまな方法があり、それゆえ、それぞれのスクリプト言語が個性を持つ。たとえば、Perl は sed や awk などの unix ツールで一般的なフィルター処理をこなすことを得意とし、そのためにいろいろな便法を提供している。近年の趨勢としては、オブジェクト指向機構の導入が一般的であり、たとえば Ruby は設計の最初からオブジェクト指向スクリプト言語を目指している。また、これはスクリプト言語に限った話ではないが、90 年代以降中間言語式のインタープリターという実装方法が多くの言語に採用されている。最も目覚しい例は Java であるが、これはスクリプト言語ではない (変数の宣言や明示的コンパイルなどがスクリプト言語らしくない特徴である)。

表 1: スクリプト言語の比較

	Perl	Python	Ruby
オブジェクト指向	△	◎	◎
多倍長整数	×	◎	◎
簡潔な文法	×	◎	○
世界的な普及度	◎	○	△

1.2.2 Python

Python は G. van Rossum が 1990 年代初頭に作り始めたスクリプト言語である。そのウェブページ [2] には「インタープリター型の, 対話的な, オブジェクト指向の」言語とある。文法は簡潔であり、モジュール機構によりかなり大規模なプログラムも作成可能である。本論文での目的である計算数論システムにおいて必要不可欠な多倍長整数が既に存在し、また、対話式インタープリターが提供される。

実行速度は中間コード式のスクリプト言語としては普通だが、別の言い方をすれば、C や Java で書いた場合よりは遅くなる。この点については、部分的に C で実装してそれを Python 側から呼び出す方法や、実行時コンパイラーの導入などにより改善が可能である。

1.3 問題点の解消

前々節 (1.1) で挙げた四つの問題点が Python の採用によっていかに解消されるか見る。

まずメモリー管理であるが、ガベージコレクションがあり、ユーザーはメモリー管理について考える必要はない。開発者も Python で書いている限りにおいてはユーザーと同列であるから、同じくメモリー管理から解放される。SIMATH でのようにメモリー管理コードを保守する必要性は全くない。

次に型の問題であるが、Python はオブジェクト指向機能を備えており、貧弱な型に悩まされる心配はない。

移植性に関しては、Python が移植されていけば良いので、我々が心配することはない。Python が移植されていくかという問題だが、現状を見る限り主要な OS である Windows, Macintosh, 各種 UNIX 系 OS に関するサポートが無くなることはなさそうである。

対話式インタープリターが最初から提供されているので、それを流用することで当面は問題無いと考えている。つまり、simcalc レベルの対話的環境を構築するのに不足はない。もし、Mathematica のような GUI 環境を提供しようと考えるならば、もう少し考えなければならないことがある。それはたとえば数式のレンダリングをどうするか、どのようなツールキットを採用すれば多くの環境で動かせるかといったことである。

1.4 その他の得失

さて、以上のように、Python の採用により最初に挙げたような問題点が解消されることが判った。ここでは、それ以外の得失について議論する。

まず、多くの人が関心を持つ速度のことだが、書かれたプログラムを実行することだけを考えれば C や Java で実装した場合に比べて確かに遅い。

これについては二つのことを指摘したい。まず、速さは純粋な実行速度だけが問題ではないということである。Python を採用することによってシステム自体の実装速度も、ユーザーのプログラミング速度も向上すると考えられる。また、実装言語とユーザーの使用言語を一致させることにより、ユーザープログラムのシステムへの取り込みが容易になり、このことも実装速度の向上に資するであろう。したがって、全体として見れば、それほど CPU 時間を無駄に使うわけではないのである。

もう一つは、それでも実行速度を向上させたいと思った時に、その道も存在するということを指摘しておきたい。一つの方法は、C で実装することである。Python から C のライブラリーを呼び出す API が存在するので、どうしても速度を向上させなければ使えないクリティカルな部分を C で実装した上でそれを Python 側から使うことができる。勿論、この方法を採用するには、Python のメモリー管理に関する知識や C でのプログラミング能力など多くのものが要求される。もう一つの方法は、実行時コンパイラーの採用である。これは標準で Python に付属するものではないが、Psyco [3] という実行時コンパイラーが開発されている。現時点では非常に残念ながら x86[2] でしか動作しないようだが、他の環境への移植が進められている。Psyco は実行時にしか変数の型が確定しない Python の特性を逆に利用して、確定した型に関してのみ実行コードをその場で生成する仕組みであり、実行速度は通常の「2 倍から 100 倍」と謳われている。[3]

[2] Intel の 8086 系統の CPU という意味で、現在では Intel の Pentium シリーズや AMD の Athron シリーズなどがある。

[3] 平均的には 4 倍程度向上するようだ

2 新システム

上で説明したような考察に基づき、新しいシステム NZMATH の構築を進めている。最新版の取得は、

http://tnt.math.metro-u.ac.jp/nzmath/

から可能である。

2.1 現状

現在[4] NZMATH の最新版は 0.0.1 である。まだまだモジュール名も関数名も変動する可能性があり、いわゆる α 版という扱いである。

含まれているモジュールを概観すると:

1. bigrandom 大きな乱数
2. euler Euler φ, Möbius 関数
3. factor 因数分解
4. gcd 整数の GCD
5. imaginary 複素数
6. integerResidueClass 剰余類 ($\mathbf{Z}/m\mathbf{Z}$)
7. lattice ラティス
8. matrix 行列
9. polynomial 多項式
10. prime 素数判定
11. rational 有理数
12. rationalFunction 有理関数
13. real 実数 (多倍長浮動小数点数)
14. ring 環
15. vector ベクトル

となっている。まだ、体に関するモジュールも楕円曲線に関するモジュールもなく、「数論」システムと名乗れる状態ではない。

これらは単純に Python のモジュールとしてインストールされるので、使い方も普通のモジュールと同じだが、Python の日本での普及度を鑑みるにこの説明では不親切なので、簡単なサンプルセッションを以下に示す。

[4] 2004 年 3 月 19 日 現在

```
$ python
Python 2.3.2 (#1, Oct 14 2003, 13:32:14)
[GCC 2.95.3 20010315 (release)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

まず注意だが、一部我々の目的にとって致命的な問題が Python 2.2 系列には存在するため、Python 2.3 以降を使わなければならない[5]。最後の >>> が Python インタープリターのプロンプトである。ここで最初に、NZMATH のモジュールを全て読み込むことにする。

```
>>> from nzmath import *
```

さて、今のところマニュアルが整備されていないので、オンラインのヘルプ機能で説明を読むと良いだろう。たとえば、rational モジュールのヘルプを見よう。

```
>>> help(rational)
```

すると、こんな画面になる。

```
Help on module nzmath.rational in nzmath:

NAME
    nzmath.rational - rational module provides Rational, Integer,
RationalField, and IntegerRing.

FILE
    /usr/local/lib/python2.3/site-packages/nzmath/rational.py

CLASSES
    nzmath.ring.CommutativeRing(nzmath.ring.Ring)
        IntegerRing
    nzmath.ring.CommutativeRingElement(nzmath.ring.RingElement)
        Integer(__builtin__.long, nzmath.ring.CommutativeRingElement)
    nzmath.ring.QuotientField(nzmath.ring.Field)
        RationalField
    nzmath.ring.QuotientFieldElement(nzmath.ring.FieldElement)
        Rational
    __builtin__.long(__builtin__.object)
        Integer(__builtin__.long, nzmath.ring.CommutativeRingElement)

    class Integer(__builtin__.long, nzmath.ring.CommutativeRingElement)
     |  Integer is a class of integer.  Since 'int' and 'long' do not
:
```

[5] 執筆時点で最新版は 2.3.3 である。

おおよそ、Unix における man コマンドの出力と同様のフォーマットで説明が出力される[6]。スペースを押すことでスクロールしていくと、class IntegerRing, class Rational, class RationalField に関する説明が現れる。それぞれのメソッドの次に書かれた説明を読むとどのような動作をするか解るはずである[7]。

元に戻って、簡単な計算をしてみよう。たとえば、有理数の足し算は

```
>>> rational.Rational(2,3) + rational.Rational(4,5)
Rational(22, 15)
```

のようになる。rational.Rational は rational モジュールの Rational という意味である。ここでは、Rational はクラスなので、呼び出しは Rational クラスのオブジェクトを生成する。足し算の記号 + が適切に多重定義されているので、有理数用の足し算関数といったものを考える必要はない。もちろん、他の演算も同様である。

以上、簡単ではあるが使い方の一端を紹介した。

2.2 将来展望

今後についてだが、2.1 節でも述べたようにまだまだ不足している機能が多いので、それを実装することが当面の目標である。

短期的目標としては、数論パッケージを名乗るに相応しい機能を早めに実装していきたい。具体的には、楕円曲線や体の定義、そしてそれらの不変量の計算方法の提供である。

中期的目標としては、使い易いユーザーインターフェイスの提供、速度の向上なども考えていきたい。

長期的には、他のシステムとの協調動作なども検討課題となるだろう。また、マニュアルの充実も図っていく必要がある。

多くの人に使ってもらえるシステムに成長すれば幸いである。

参 考 文 献

[1] Matsui, Kobayashi, Abe, Nakamula, "SIMATH — Recent Developments in TMU" in Cohen A.M. et al (ed.) *Mathematical Software: Proceedings of the First Congress of Mathematical Software* pp.505-506, World Scientific, August 2002.

[2] *What is Python?*, http://www.python.org/doc/Summary.html

[3] *Psyco Homepage* http://psyco.sourceforge.net/

[6] これは pydoc という独立したコマンドでシェルから呼び出すこともできる

[7] これらはソースコードに埋め込まれた文字列であり、その品質については我々開発者がどれだけ説明を書くことに力を注ぐかに懸かっている。